EXPERIENCE WITH THE DATA ARCHIVER IN DOOCS

Harald Keller, Olaf Hensler, DESY Hamburg, Germany

ARCHITECTURE

At the Tesla Test Facility (TTF) more than 20,000 data channels are permanently archived. Because of that huge amount an important design goal of DOOCS was the implementation of a device server as an independent program that completely controls a number of devices and provides the device data to the network and receives messages from the clients. The implementation of the client communication requires only a few calls - two different calls to read data, one to send data and another to query the actual list of devices and properties. The implementation of the list query as hash search reduced the search time compared to the previous sequential search significantly.

INTRODUCTION

In order to be independent from the Ethernet, a harddisk is attached to every server station. This allows booting and running individual servers without the network. Because of the local harddisk, archiving history data can be done by the server as well. This reduces the network traffic as well and allows analyzing problems that happened during network breakdown without loosing data. Therefore each DOOCS server has its own archiver to store and retrieve the filtered data in a history file with one ring buffer per channel [*fig. 1*]. The archiver library provides three different archive formats.

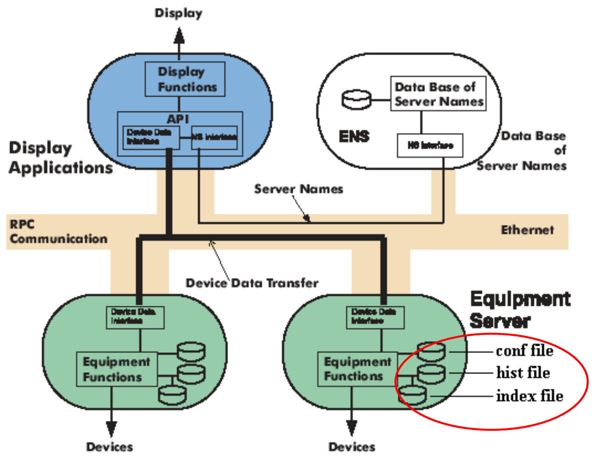


Figure 1: DOOCS client server structure.

The *TDS* format stands for a structure consisting of a ***T****imestamp*, a float ***D****ata* value and a ***S****tatus* entry. This archiver has two ring-buffers for every channel [*fig. 2*], one fast memory buffer to see the noise of the signal, no filtering is done except when the value is equal to the one before. The second one is the buffer inside the archive or history file for medium speed archiving. It exists in two versions, without or with an internal index. Writes to the ring buffer are filtered to reject noise from filling up the storage file. The filter parameters are online adjustable.

The threshold values in filter 1 can be an absolute difference, a difference in mask bits or a difference in percentage. With the last type one can also set a minimal absolute threshold and a time limit after that a value has to be stored in any case [*table 2*].

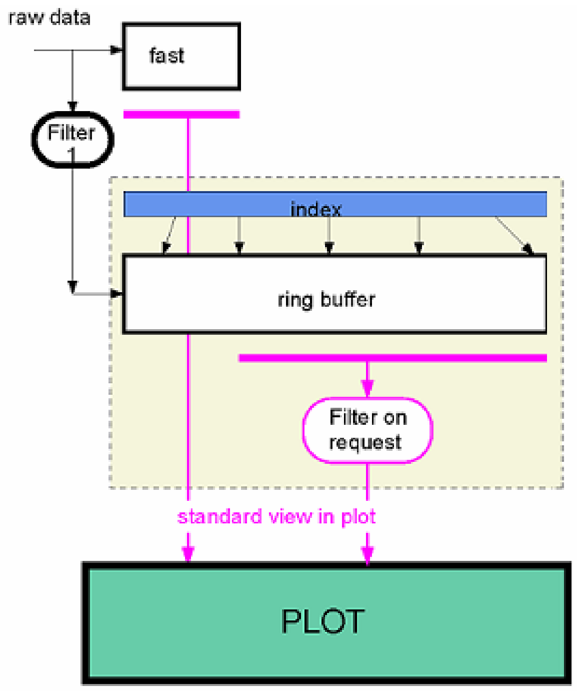


Figure 2: Memory and file ring buffer.

Both ring-buffers are running in parallel to each other getting the same data values. To readout one of the buffers, one has to provide the TTII data-type in the get call. The first integer value switches between memory and file buffer:

- 0: -> memory buffer at the beginning and then file buffer for the remaining time
- 1: -> only file ring-buffer.

In this paper the focus is on the TDS archive format.

The second archive format is the *comment* string, which is a 80 character string array. This is useful to provide additional logging on a per device basis.

The third type of archiver stores complete *spectra* as one entry in a ring buffer, which is essential a 2,048 item float array. With this function one can archive scope pictures or mass spectra, for instance. Up to 100 spectra can be stored for one channel. One can retrieve a list of stored spectra in a given time period in order to choose a spectrum from the archiver file.

Furthermore every server program keeps a local data base (config file) of its actual configuration state in order to restart the system unchanged after a power failure. Since this data base is a file on the local disk, in this case too a server does not depend on the network.

BY EXPERIENCE TO NEW FEATURES

Archive or hist files with a huge amount of device histories in one single file led to very long startup times of a DOOCS server. By the implementation of a separate **index file** [*fig. 1*] the startup times could be reduced to about 25% of the original times. From the remaining time approximately 70% is used to read and interpret the configuration file. Here surely further improvement could be achieved.

During the display of very long ranges of history data it could happen in the past that there where some timeouts in the network traffic while retrieving the data from the hist file. The network packages are limited to 2,000 records and the search algorithm in the ring buffers had to be much faster. By the implementation of a **RBIIndex** (ring buffer internal index) [*fig. 2, 3*] the data retrieval time could be reduced massively. The start position for the read operation is quickly found by the help of the index and for one network package of 2,000 records no more than 4 logical parts of a ring buffer have to be read.

The size of a RBIIndex depends on the ring buffer size. It is calculated once during the creation. A hist file consists of

- one mainheader containing the file type version, device count and the byte order (big / small endian) to ensure a correct use on the compatible computer system (PC or Sun)
- one or more ring buffers which consist of
    - RBIIndex (if it is version 1)
    - History data entries

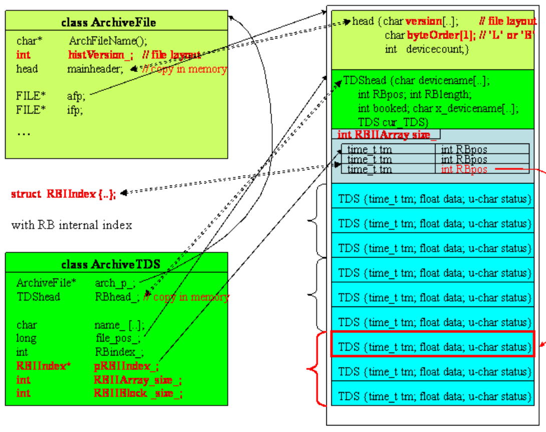


Figure 3: Data retrieval by the internal index.

The problems that appeared in the course of time during the handling of hist files led to the idea better to have **more different hist files** with less ring buffers instead of one big hist file with a lot of ring buffers. For example, it is much more difficult to delete one or more ring buffers and you have to keep others if there is a big hist file with a huge amount of devices. Performance tests even showed an advantage of this architecture, and the previously implemented separate index file is no longer needed.

We developed the utility **histOne2n** that separates a big hist file with a lot of ring buffers or devices into many different smaller hist files ("one to n" files) by taking the location names to create the new hist file names automatically [*fig. 4*]. The utility also makes it possible to increase the size of the ring buffers during the copy.

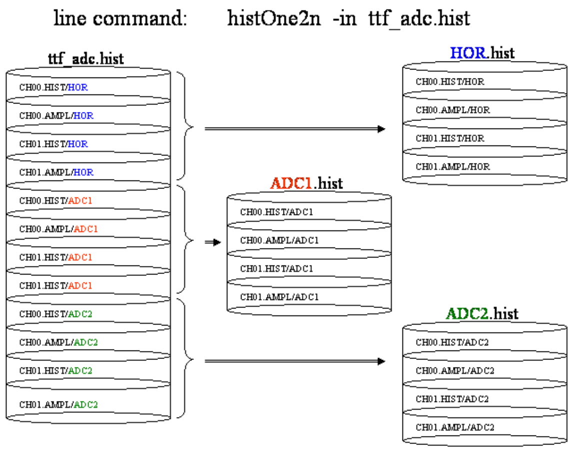


Figure 4: histOne2n utility.

In the last year server programmers used this utility to split their hist files. The newly created smaller files automatically obtain ring buffers with a RBIIndex.

The statistical overview about the **usage** of all ring buffers [*fig. 5*] shows that the main part of the ring buffers is filled up to equal or less than 10%. About a third is filled up to equal or less than 100%. In these cases it is the question whether the ring buffer length is big enough to keep enough data from the very past.

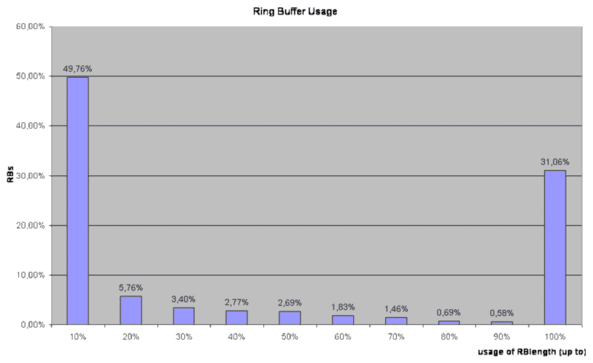


Figure 5: Ring buffer usage.

But as one can see in *figure 6* too many ring buffers are defined much too small. A **size** of less than 10,000 records does not seem to be reasonable as known from experience. The existing smaller ring buffers might remain from server test runs. After testing the programmers sometimes forget to change the size.

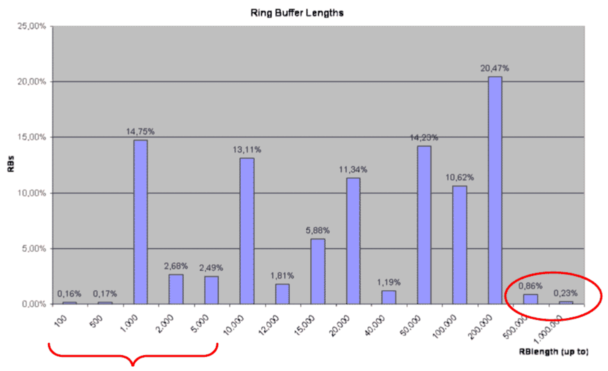
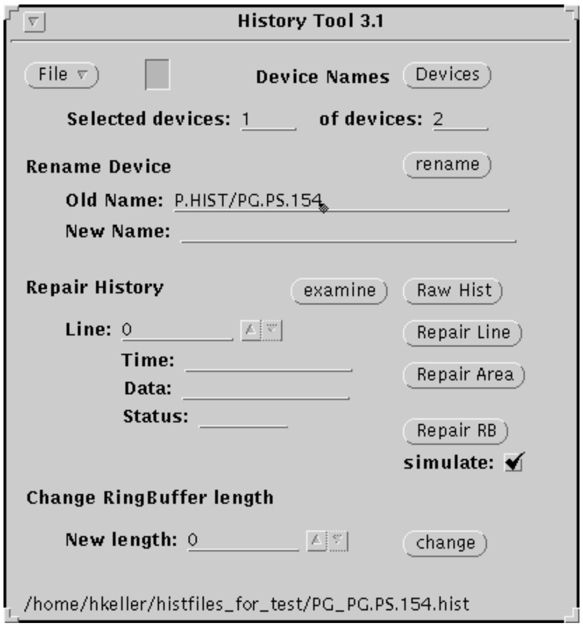


Figure 6: Ring buffer lengths.

A **History Tool** (**xhisttool**) for maintaining history data and hist files [*fig. 7*] was developed. This tool allows displaying and modifying the data of an archiver. It is used for

- debugging and error correction
- for statistical investigation
- increasing ring buffers
- deactivating ring buffers by renaming them to "UNUSEDx" where x is a digit to prevent from using a name twice what is forbidden


History Tool 3.1

File | Device Names | Devices

Selected devices: 1   of devices: 2

Rename Device | rename

Old Name: P.HIST/PG.PS.154

New Name:

Repair History | examine | Raw Hist

Line: 0 | Repair Line

Time: | Repair Area

Data:

Status: | Repair RB

simulate: ✓

Change RingBuffer length

New length: 0 | change

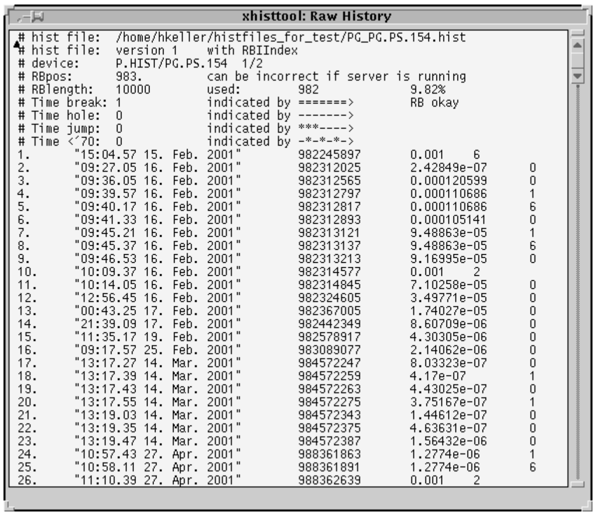
/home/hkeller/histfiles_for_test/PG_PG.PS.154.hist

Figure 7: xhisttool.

The utility has the following functions:

Devices	Display a list of all devices in a hist file. Here you select one or several devices you want to handle.
rename	Rename of one single device. This function prevents from double use of names which would mean to have a 'dead' unused ring buffer as happened in the past.
examine	Examine all ring buffers of the hist file. You get a good overview of - the structure of a hist file - the size and the usage of the ring buffers - possible chronological errors
Raw Hist	List information about a file, the ring buffer (from the header), some statistics and the contents of the ring buffer data. They are in a physical and not in a chronological order. During reading this order is inspected and possible errors will be indicated. [*fig. 8*]
Repair Line	Manual reparation of the values of a single entry
Repair Area	Manual reparation of the values of a range of entries
Repair RB	Automatic reparation of a complete ring buffer
change	Copy the complete hist file to a NEWLENGTH file. Xhisttool increases the ring buffer lengths of all previously selected devices. The other not selected devices keep their original ring buffer lengths. Ring buffers with a device name "UNUSEDx" are not copied.

Table 1: xhisttool functions



```
# hist file:  /home/hkeller/histfiles_for_test/PG_PG.PS.154.hist
# hist file:  version 1    with RBIIndex
# device:     P.HIST/PG.PS.154  1/2
# RBpos:      983.         can be incorrect if server is running
# RBlength:   10000        used:          982           9.82%
# Time break: 1            indicated by =======>        RB okay
# Time hole:  0            indicated by ------->
# Time jump:  0            indicated by ***---->
# Time <´70:  0            indicated by -*-*-*->
1.       "15:04.57 15. Feb. 2001"      982245897        0.001     6
2.       "09:27.05 16. Feb. 2001"      982312025        2.42849e-07       0
3.       "09:36.05 16. Feb. 2001"      982312565        0.000120599       0
4.       "09:39.57 16. Feb. 2001"      982312797        0.000110686       1
5.       "09:40.17 16. Feb. 2001"      982312817        0.000110686       6
6.       "09:41.33 16. Feb. 2001"      982312893        0.000105141       0
7.       "09:45.21 16. Feb. 2001"      982313121        9.48863e-05       1
8.       "09:45.37 16. Feb. 2001"      982313137        9.48863e-05       6
9.       "09:46.53 16. Feb. 2001"      982313213        9.16995e-05       0
10.      "10:09.37 16. Feb. 2001"      982314577        0.001     2
11.      "10:14.05 16. Feb. 2001"      982314845        7.10258e-05       0
12.      "12:56.45 16. Feb. 2001"      982324605        3.49771e-05       0
13.      "00:43.25 17. Feb. 2001"      982367005        1.74027e-05       0
14.      "21:39.09 17. Feb. 2001"      982442349        8.60709e-06       0
15.      "11:35.17 19. Feb. 2001"      982578917        4.30305e-06       0
16.      "09:17.57 25. Feb. 2001"      983089077        2.14062e-06       0
17.      "13:17.27 14. Mar. 2001"      984572247        8.03323e-07       0
18.      "13:17.39 14. Mar. 2001"      984572259        4.17e-07          1
19.      "13:17.43 14. Mar. 2001"      984572263        4.43025e-07       0
20.      "13:17.55 14. Mar. 2001"      984572275        3.75167e-07       1
21.      "13:19.03 14. Mar. 2001"      984572343        1.44612e-07       0
22.      "13:19.35 14. Mar. 2001"      984572375        4.63631e-07       0
23.      "13:19.47 14. Mar. 2001"      984572387        1.56432e-06       0
24.      "10:57.43 27. Apr. 2001"      988361863        1.2774e-06        1
25.      "10:58.11 27. Apr. 2001"      988361891        1.2774e-06        6
26.      "11:10.39 27. Apr. 2001"      988362639        0.001     2
```

Figure 8: Listing by Raw Data function.

A part of the functionality used in the xhisttool GUI can be used also as a line command in the **doocshisttool**. Furthermore it is a good tool if you want to move a hist file from a Sun to a Linux system or vice versa. Because of the different byte orders of the different computer systems (little and big endian) the hist files are temporarily converted into ASCII files.

The script **histstat.csh** delivers statistical data of all hist files of all servers based on the listing of the examine function and prepares them for **Excel** use.

FUTURE PROSPECT AND CONCLUSION

Because data get overwritten when a ring buffer is full, it could be useful in some cases to have a third buffer for **long time archiving** [*fig. 9*]. First we have to decide what kind of compression and filtering we want to use for this feature. **-** On the other hand there is a big potential by selecting an appropriate size for a ring buffer. Too many ring buffers are defined too small. About 20% of them are under the suggested minimum size of 10,000 and only 1% uses the larger sizes of 500,000 or 1,000,000 entries.

The idea is to keep the existing ring buffer files, filter (Filter 2) and transfer the data entries that would be deleted to one or more separate sequential files. These long time archive files should have their own index for faster data retrieval. A supplementary overview file, that is automatically written or can be separately created on demand by special filtering (Filter 3), should offer an easier and faster access to the wanted data.

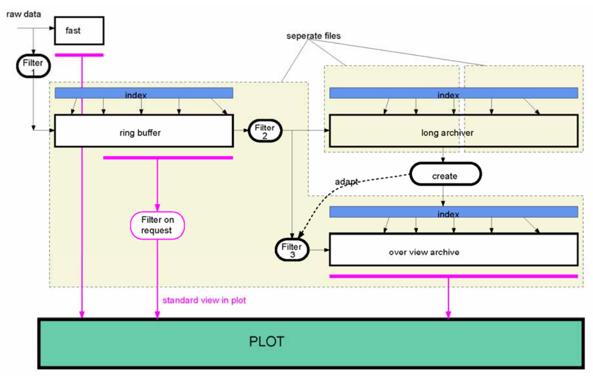


Figure 9: Long time archiver.

Furthermore there is a need to **set Filter 1** [*fig. 2, 9*] **automatically**. At the moment it gets default values (50% difference – IFFF = 0 0.5 0 0) during the first start of a server program. Of course these values are not ideal for every type of channel. But as these parameters are not changed by the users in general, an intelligent automatic filter setting could develop more reasonable values.

Filtering is controlled by the IFFF (int, float, float, float) structure. The idea is to begin at the first startup with an *automatic filter* (I = -1) which investigates all unfiltered data of the memory ring buffer. When this buffer is filled up an appropriate filter is set and will be kept (I >= 0) until it is a manually changed.

Table 2: Archive filters

I (filter type)		**F** (value)	**F** (value)	**F** (value)
-1	automatic			
0	percentage	difference	min diff	time limit
1	absolute	difference		
2	mask	mask bits		